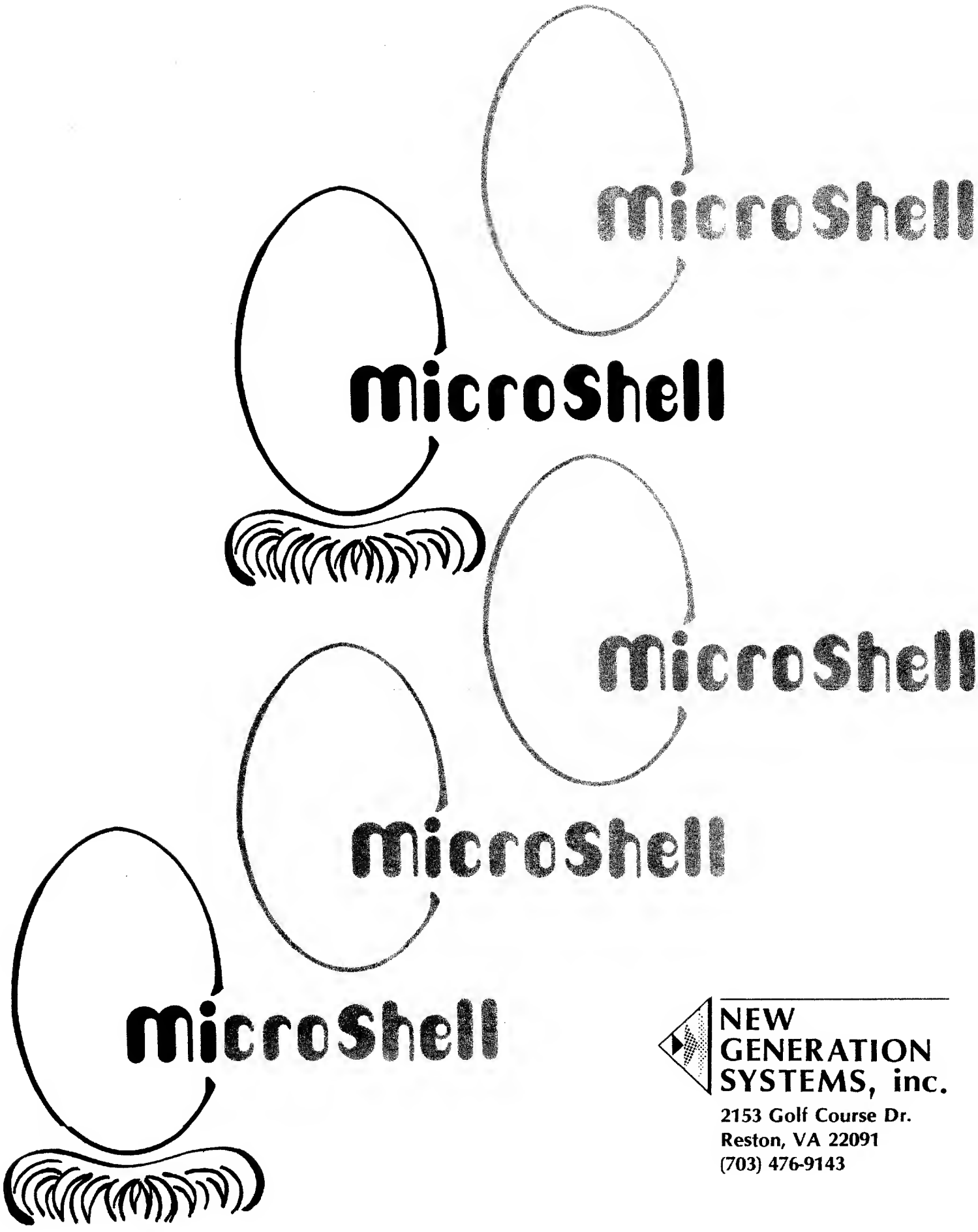

NEW
GENERATION
SYSTEMS, inc.
2153 Golf Course Dr.
Reston, VA 22091
(703) 476-9143

M I C R O S H E L L

UNIX Features for CP/M

User's Manual Version 1.21

May 30, 1982

New Generation Systems, Inc.
2153 Golf Course Drive
Reston, Va. 22091
(703) 476-9143

Summary of Changes in MicroShell Version 1.2 and 1.21
May 30, 1982

A number of significant enhancements have been added to the current version of MicroShell while its size has been reduced by one page (0.25 K bytes) to 9.25 K bytes.

The following is a brief summary of the changes:

- Input/Output Redirection: Direct BIOS calls supported
- Input Redirection: Added "CP/M" and "UNIX" modes. Added "Transparent" flag.
- Verbose Flag: Now causes all commands and input from a file to be echoed.
- Output Redirection: User input to a program can now be redirected.
- Redirection and the Printer: Added redirection to the printer and redirection of printer output to the console.
- Repeat of Previous Command capability
- New command "TYP": Paged file display
- User adjustable delays for running program demonstrations with MicroShell.
- CP/M error conditions no longer cause exit from MicroShell.
- Command (shell) file changes
- Abbreviated shell flag display
- Miscellaneous bug fixes.

A more detailed discussion of each change now follows:

1. Input/Output Redirection: Previous versions did not redirect input or output which bypassed the CP/M BDOS and went directly to the BIOS. All I/O can be redirected now, not just I/O that goes through the CP/M BDOS. So programs like Microsoft Basic and Ashton Tate's dBase II, which do direct BIOS I/O, can now be redirected.

2. Input Redirection: Input redirection is one of the most difficult UNIX features to implement under CP/M due to the many "ways" a program can get input from CP/M, how to recognize the end of the input and what to do at the end of the input. We have

established two input redirection modes, set by the "M" mode flag:

Mode "On" = CP/M mode: Input returns to the keyboard after the end of the shell or input file. This is the default mode and is the way "submit" works under CP/M. This mode permits, for example, a file which could contain a number of initial Wordstar commands (margins, help levels, tabs, etc) and then return the input to the keyboard so the user can begin editing.

Mode "Off" = UNIX mode: At the end of the input file, i.e. when a control Z is read from the file, a control Z followed by a carriage return is returned to the program, which is then responsible for properly terminating. This mode permits a program to operate on normal text files and receive a special signal at the end of the file. This protocol appears to be compatible with some of the "software tools" packages that are on the market.

Argument substitution (e.g. $1), control character substitution (e.g. ^Z) and escape character processing (e.g. \$) still are done in either input mode. In addition, line feed gobbling may be occuring depending on the "G" gobble line feed flag state. For those who can't live with this for some special situation, there's a way to tell MicroShell to just pass the characters through with no action - the new "T" Transparent flag. If the "T" flag is set "ON", no special character recognition is done in the input redirection process and the characters will be passed "raw" to the program until the physical end of file is reached. The physical end of file is not the control Z (which is the logical end of file) but the end of the last sector assigned to the file by CP/M. Action at the physical end of file depends on the input mode ("M" flag); CP/M mode returns to the keyboard and UNIX mode sends ^Z followed by carriage return.

3. The Verbose flag is now default "On" and only causes lines of commands read from shell files or redirected input from a file to be echoed. These lines or characters taken from an input file were not previously echoed. Commands given directly to the MicroShell prompt are now not echoed again.

4. When output is being redirected to a file, input to a program typed by the user is now also redirected to the file.

5. Printer Redirection:

a. Redirection to the printer: Output can be redirected to the printer with:

```
program >$P     [program output goes only to printer]

     or

program >+$P    [to echo output to screen]
```

b. Redirection of printer output: Output from the printer can be redirected to a file with the following commands:

```
program >*filename [printer output goes only to file "filename"]

      or

program >*+filename [printer output goes to file "filename" and
                     to the printer]
```

This feature is useful in capturing a report which the issuing program only sends to the printer or in debugging printer setup code in programs.

5. The previous command may be repeated by entering an exclamation point followed by a carriage return. If a shell file was the last command, MicroShell will repeat the last command of the shell file - not reexecute the whole shell file. Though a number of users had asked for the capability to edit the last command, that code is presently too big to fit with our goal to reduce the size of MicroShell.

6. Paged File Listing. A new command is recognized by MicroShell - "TYP". "TYP" is the same as "TYPE" except that the output is stopped every 23 lines until a character is typed.

7. There are three variable delays now built into MicroShell to permit MicroShell's action to be slowed down when running a demonstration shell file and/or for other special purposes. One delay is in the routine which returns a line of input (BDOS call 10) during input redirection, one delay is in the routine which returns a single character (BDOS call 1, 6, or direct BIOS call) during input redirection and the third delay is just prior to printing the command prompt. (This delay is specifically for Heath systems that lose the prompt coming out of a program that resets the video driver - like the Pie editor.) These delays must be patched with DDT or a program may be written which sets them. All three are in 1/10ths of a second (for a 2 MHz 8080) and are set to a default value of 0. When "sh.com" is loaded with DDT their locations are:

```
Single character delay: 2DB6 Hex
Line Input delay:       2DB7 Hex
Prompt delay:           2DB8 Hex
```

Therefore to set a 2 second line input delay, set 2DB7 to 20 (decimal). Heath users should set the prompt delay to 0.1 sec.

8. All CP/M error conditions are now trapped in MicroShell. After a CP/M error, MicroShell issues a "-L" command internally to relog the disks and returns to the prompt. If a ^C is entered after the MicroShell prompt, this action also occurs. Note that this has the same functional effect as typing a ^C to the CP/M prompt but MicroShell's action is faster than a normal CP/M warm start. (One of the side effects of "catching" ^C's is the

somewhat strange screen action after a ^C is typed - a carriage return followed by "^C" overprinting the prompt. This is normal in the method MicroShell must use to "catch" the ^C.)

9. Command File Changes:

a. Some programs erase "$$$.sub" on drive A if they exit abnormally to cause a submit file to be stopped. The BDS C compiler is an example. MicroShell also now recognizes this to terminate a shell file (or a multiple (semicolon) command) if in progress. This feature - stopping shell files when a program erases a "$$$.sub" file - is automatically disabled when "submit" is run because submit also erases "$$$.sub". It is sometimes desired to use "submit" to build a "$$$.sub" file to execute upon exit from MicroShell. The feature is also inhibited for an explicit "era $$$.sub" issued from a shell file. For the shell file to be terminated, an executing program must erase "$$$.sub". The clever user will see how to exit a shell file on some desired condition by using this feature.

b. For "submit" compatibility, MicroShell now recognizes either ":" or ";" in the first column of a line in a shell file to signify a comment line. Also, MicroShell now ignores the command "XSUB" so that all submit files should run fine under MicroShell.

c. When operating from a user number other than 0, Micro-Shell now will search user 0 of the current disk for a shell file before beginning to look for ".com" files. This permits common shell files to be kept only on user 0 of a disk rather than in each user area. If the command is preceeded with a disk drive, MicroShell first looks in the current user # of the disk drive specified and then in user 0 of the disk drive specified for a shell file before looking for ".com" files. It is possible to create a link to a library of shell files by putting a simple shell file, say "cmd.sub" in user 0 of each disk drive. "cmd.sub" would contain:

```
a:$1 $2 $3 $4 $5 $6 $7  (for shell file library on "A")
```

Then any shell file on drive A can be accessed by typing:

```
cmd [shellfile name] [args]
```

d. TIMBUS A new program - "TIMBUS.COM" - has been added to the distribution disk specifically to support dBase II users using the "quit to" command. It may also be of use to others. It would normally be used in a shell file and performs the following action:

(1) Opens "$$$.sub" on the current drive (and user area) or if not found, on drive A in the current user area.

(2) Outputs the commands contained in the "$$$.sub" file in reverse order (last first) to the console.

(3) Renames "$$$.sub" to "--shtmp" and then erases it (so that shell file is not terminated by erasing "$$$.sub".)

So here's how you would use "TIMBUS" with dBase II. Edit a command file to run dBase II such as "rundbase.sub" which will contain:

```
dbase $1 <$T [to run dBase and take input from keyboard]
timbus >temp.sub  [to write dBase II "quit to" cmds to
                        temp.sub]
temp
```

Thus, MicroShell can continue to execute the dBase II "quit to" commands even though MicroShell does not warm boot CP/M to execute the "submit" ("$$$.sub") file that dBase II writes.

10. In reducing the size of MicroShell while adding a number of features, something clearly had to go. One of the things that went was the long listing of the Flag Meanings when "-S" was typed. The flags are now labeled very tersely as follows:

ABBREV	Meaning	Default Value
FS	File Search	On
GL	Gobble Line Feeds	On
UC	Upper Case Command Line	On
MD	Mode for Input Redirection (On=CP/M//Off=UNIX)	On
VB	Verbose	On
TR	Transparent Input	Off

11. Miscellaneous Errors Corrected:

a. ">>" with an explicit disk drive given would not work. Fixed now.

b. A minus sign after a pipe symbol set the console status on the left side of the pipe symbol instead of the right side. This has been fixed.

c. Programs that set the disk drive directly to the BIOS left MicroShell confused about which drive it was on. This has been fixed we believe.

d. When MicroShell ran programs larger than one extent (16 or 32 K depending on the disk density) or attempted to load overlays larger than one extent from a user area other than 0, they would sometimes bomb. This has been corrected.

e. One program from the CP/M Users Group would not run with MicroShell - "SD" - which was a directory program. This program alters error vectors inside of the CP/M BDOS. These error vectors are not documented in the standard CP/M documentation and are not supported by MicroShell. "SD" will run if it is reassembled with "DOPT" set FALSE to prevent "SD" from altering the error vectors.

12. A "BAD LOAD" error message has been added. If a program is too large to be loaded under MicroShell, this message will be printed and MicroShell will return to the command prompt.

13. In the revision 1.1 change notes, the patch point for changing the number of columns displayed for the "dir" directory command was given so that Apple users with only 40 columns and Osborne users could keep the directory display on the screen. This patch point is now 2D8E Hex when MicroShell is loaded with DDT. After the number of columns has been changed as desired, the new "sh.com" should be saved with "save 46 sh.com".

14. Revision 1.21 changes:

a. The scheme of printer redirection to a file was changed to use the >* command line scheme instead of the shell flag in revision 1.2. In addition, only printer output now goes to the file instead of being merged with console output.

b. When programs taking console input directly from the BIOS (like Wordstar, Mbasic and dBase II) are run from a shell file or with input redirection, MicroShell now does not echo the input to the screen as the program itself does. This had resulted in double characters on the screen when the Verbose flag was on.

c. In MicroShell 1.2, direct BIOS disk select calls were rerouted to the BDOS to correct the problem of MicroShell and the BDOS not knowing that a program had changed the disk drive. This solution caused problems with formatting programs. In revision 1.21, this situation has been improved though a "-L" may be necessary after running "format" or "copy" programs which directly access the BIOS drive select.

d. Handling of the BDOS call 6, direct console I/O with redirection was corrected.

e. The remaining change in Revision 1.21 was to add the Mode flag to the CUSTOMIZ program to permit changing its default value.

f. A summary of the additional MicroShell special characters and commands not listed in Appendix E of the manual:

Special Characters

Char	Meaning	Example
>>	Append Console Output to file	stat >>filename
>>+	(with echo to console)	dir >>+filename
>*	Redirect Printer Output to file	ws >*filename
>*+	(with echo to printer)	ws >*+filename
$P	Printer (Redirection to printer)	stat *.* >$P
	(with echo to console)	stat *.* >+$P
^C	Control C typed on command line is identical to -L flag	% ^C [MicroShell logs in disks] %
!	Repeat previous command	% copy all [copy executes] % !
	[MicroShell echoes last command]	copy all [copy executes again]

MicroShell Flags

+M	Sets CP/M Input Mode (return input to keyboard at end of input or shell file) (Default)
-M	Sets UNIX Input Mode (send ^Z and carriage return to program at end of input or shell file)
+T	Transparent Mode On; MicroShell ignores all special characters ($,^,\) during input or shell files.
-T	Transparent Mode Off (Default)

Summary of Changes in MicroShell Version 1.1
February 24, 1982

1. The amount of memory that MicroShell takes away from the CP/M Transient Program Area (TPA) has been reduced to 9.5 K bytes.

2. Some programs that used the default disk location at 0004 did not work properly with MicroShell. The symptom was that the wrong drive was made the default drive. This has been corrected.

3. The number of columns in the "dir" directory listing was made a variable and can be changed from the CP/M default of 4 columns. This may be useful for narrower screens than 80 columns. (Apple, etc.) In fact, users with 80 column screens can change it to 5 columns if desired. It has not yet been added to CUSTOMIZ but is found at location 2E94 hex if MicroShell is loaded with "DDT". After setting this location to the desired number of columns for "dir" to display, the new MicroShell image should be saved with "save 47 sh.com".

4. CP/M "system" files were being shown in the "dir" directory listing. This has been corrected and they no longer show unless a "-s" flag follows "dir" in the command line. For example: "dir -s *.*" will show all files in the current user area, including "system" files.

5. MicroShell was very intolerant of spaces after the ">" and "<" redirection operators. If spaces were present, garbage files were sometimes created. This has been corrected and "white space" -- spaces or tabs -- are now optional between the ">" or "<" and the file name. Note that no white space is allowed between a ">" or "<" and a "+" or "-" modifier.

6. Double digit user numbers, if displayed in the prompt, could cause MicroShell to bomb. This has been corrected.

7. See Section 3.6.7 of the manual for proper MicroShell operation with the Wordstar word processing program.

8. Heath Users: When using the "Pie" editor from the Software Toolworks with MicroShell, the user may notice that the first prompt which MicroShell issues after exiting "Pie" is missing some of the first characters. This is caused by the fact that "Pie" issues a reset command to the video driver just prior to exiting to CP/M. This reset command requires a relatively long time for the video processor to complete (compared to other functions.) Thus, when MicroShell tries to print its prompt, the processor is still busy with the reset and misses some of the first characters in the prompt string. This does not cause a problem under CP/M alone because CP/M's warm start also takes some time to complete. Since there's no warm start when running under MicroShell, this can be solved in two ways. The first method is to just hit return, issuing a blank command line; MicroShell will just reprint the prompt. The second method is to make a custom prompt (Section 2.4.5) that includes about 10 leading blanks before the "%n" to eat up some time and allow the reset to finish.

COPYRIGHT NOTICE

TRADEMARKS

The following trademarks are referenced throughout this manual:

ACT 80, ACT 86, PASCAL/M	Sorcim Corporation
CompuPro	Godbout Electronics
CP/M, CBASIC, MAC	Digital Research Corporation
Spellbinder	Lexisoft, Inc
UNIX	Bell Telephone Laboratories
Wordstar, Spellstar	MicroPro International Corporation

How to Run MicroShell without Reading the Manual

Everyone is always anxious to run a new program without having to read through the manual first. Here's how to run MicroShell.

1. **Copy the distribution disk before you do anything else.** It should have the following programs on it:

SH.COM	MicroShell
CUSTOMIZ.COM	MicroShell Customization Program
ECHO.COM	A useful program for shell files which merely echoes its arguments to the screen (for messages from shell ("submit") files.)
FULLPRMP.SUB NORMPRMP.SUB	Demonstration shell files which will change the prompt.

2. With CP/M running in your computer, type "sh", followed by a carriage return of course.

3. MicroShell will sign on with a "%" prompt (like UNIX).

4. Give MicroShell a few CP/M commands like "dir" and "stat" to assure yourself that it's working.

5. Now try one of MicroShell's features; type:

```
dir >files
```

You shouldn't see anything but the disk will click as MicroShell puts the directory listing into the file "files". Look at "files" by typing:

```
type files
```

There's your directory! Now let's try another MicroShell feature, multiple commands. Type:

```
dir >>files;type files
```

Now you see a second directory "appended" (the ">>") to the end of the first. And the second half of the command line executed as soon as the first was done!

6. Now try this. Type:

```
fullprmp
```

Now you've got a new prompt that includes the disk drive, the user number and a bell (to tell you when a command is done.) (If you got a "fullprmp?", you haven't copied the ".sub" files on

the distribution disk to the disk you're using. Go back and do that and then continue.)

If you don't like that prompt, type:

normprmp

and you get the "%" prompt back. You can change the prompt to your liking. See Section 2.4 on "Shell Flags."

7. Now you have the beginning of MicroShell and an idea of the power of the UNIX operating system that it emulates under CP/M. A summary of the special command line characters and shell flags follows. Try each feature to get an idea of what MicroShell can do. This summary is identical to Appendix E and can be removed from the manual and used for reference.

Summary of MicroShell Commands

Special MicroShell Characters in Command Line

Char	Meaning	Example
>	Output Redirection	stat >filename
<	Input Redirection	ed file <script
¦ ^	Pipe output to input of next cmd ("^" and "¦" are equivalent)	prog1 ¦ prog2 ¦ ... stat *.*¦pip lst:=con:
^	"^" in shell and input files causes next character to be its control equivalent.	^C (or ^c) changed to 03
:	When first character on a shell file line, causes line to be treated as a comment (ignored).	: this is a comment
+	Echo redirected Output to Console	stat >+filename prog1 ¦+ prog2 ¦ ...
-	Return "character ready" to console input status calls	sysgen <-script prog1 ¦- prog2 ¦ ... prog1 ¦+- prog2 ¦ ...
;	Separate commands	era *.bak;stat;ed test <script
"	Treat arguments with embedded spaces or tabs as 1 argument and ignore special characters inside quotes	echo "This is one argument"
\	Ignore special meaning of next character	dir \>file (filename ">file")
$	Argument substitution in shell (command) files (0-19)	comfile test data if "comfile.sub" contains: pip b:=a:$1 pip b:=a:$2 then MicroShell executes: pip b:=a:test pip b:=a:data
$T	Redirect Input back to console in a command file	pip b:=a:$1 b:=a:$2 ddt <$T

Shell Flags

Flag	Meaning	Example
+f or +F (Default)	Auxiliary file search enable	% +F (Auxiliary file search on)
-f or -F	Auxiliary file search disable	% -F (Auxiliary file search off)
+g or +G (Default)	Gobble line feeds during Input redirection	% +G (Line feeds removed from input)
-g or -G	Don't gobble line feeds during Input Redirection	% -G
-l or -L or +l or +L	Login current disk (after changing disks)	% -l % (new disk logged into CP/M for writing)
-p or -P or	Prompt string Uses "C"-like format: % - Next char special (%% gives %)	-p "%n%%" gives: (CR, LF)%
+p or +P	n/N - Newline (CR, LF) d - Lower case drive D - Upper case drive u/U - User number	-p "%nDrive:%D User%U %" gives: (CR, LF)Drive:A User:0 %
-s or -S or +s or +S	Shell Status report (Shows status of flags)	% -S File search: On Gobble lfs: Off Ucase cmd: On Verbose: Off
+u or +U (Default)	Upper case translation of command line (like CP/M)	% +U % echo this is upper case THIS IS UPPER CASE
-u or -U	No case translation on command line (allows passing lower case command line to a program)	% -U % echo this is lower case this is lower case
+v or +V	Verbose mode: Echo commands before execution	% +V (All commands echoed) comfile test data pip b:=a:test pip b:=a:data
-v or -V (Default)	Disable Verbose mode	% -V (No echo of commands)
-x or -X	Exit MicroShell and return to CP/M	% -x A>

Table of Contents

Table of Contents (Cont)

1.0 Overview

MicroShell is a CP/M program which adds powerful, user-friendly capabilities to the CP/M operating system similar to many of the functions available in the UNIX operating system. Compatibility with existing CP/M software is retained while adding the UNIX features to the operation of existing CP/M software. New software applications and tools can be designed and implemented with much less effort using the features available in MicroShell. MicroShell can be tailored to a user's system and experience level, providing additional information and help for a new user or crisp, elegant power for an experienced user.

1.1 System Requirements:

MicroShell requires CP/M 2.2, at least 32 K of memory and at least one disk drive.

1.2 Summary of MicroShell Features:

The new user of MicroShell should first read the Preface -- How to Run MicroShell without Reading the Manual, following the examples by actually using MicroShell. After he has gained an initial familiarity with MicroShell, Basic MicroShell Commands, Section 2.0, will provide more details on using MicroShell.

2.0 Basic MicroShell Commands

This section will cover the basic MicroShell Commands, including:

2.1 - Executing MicroShell
2.2 - CP/M-like functions
2.3 - Forming MicroShell command lines
2.4 - MicroShell Flags
2.5 - Output Redirection
2.6 - Input Redirection
2.7 - Pipes

2.1 Executing MicroShell

The MicroShell program is named "sh.com" and is executed by typing:

```
sh

or

sh [ initial command line ]
```

from the CP/M prompt ("A>", if the user is logged into CP/M on the A disk drive.) MicroShell will respond with:

```
MicroShell Version X.X
Copyright (c) 1981 New Generation Systems

(If an initial command line was given, the command
 is executed prior to the initial % prompt.)

%
```

If the user has altered the default prompt from the "%" in the delivered MicroShell, the custom prompt is displayed.

The initial command line may be any legal MicroShell command. For example, if the user wanted to log into drive B immediately, he could execute MicroShell with the following command line:

```
sh b:
```

MicroShell would then change the default drive to B prior to displaying the initial prompt. If a user desired to set some of the shell flags (Section 2.4) on initial entry, a shell file (Section 4) could be executed on initial entry to perform these functions. If the file, "init.sub" contained the following lines:

```
+V
b:
dir
```

then by typing:

```
sh init
```

MicroShell would turn on the Verbose flag (Section 2.4.8), change the default drive to B, and display the directory prior to issuing the initial prompt. The shell flags and shell files are fully described in Sections 2.4 and 4 respectively.

Before we proceed to discuss MicroShell's other features, here's how you get out of MicroShell and back to CP/M. Just type the -x (or -X) "eXit" shell flag, by itself, in response to the MicroShell prompt. MicroShell will exit back to CP/M and CP/M will display its prompt for the current drive, e.g.:

```
% -x        (<---user types "-x", carriage return)
A>          (<---CP/M prompt)
```

2.2 CP/M-like Functions

MicroShell, on initial entry, relocates itself just below the CP/M Basic Disk Operating System (BDOS), replacing the CP/M Console Command Processor (CCP). The CCP is the part of CP/M which interprets commands by the user and performs the built-in functions: DIR, ERA, REN, SAVE and TYPE. The vast majority of user programs - word processors, compilers, accounting programs, etc. - actually overlay the CCP to use its space in memory for their own use. MicroShell replaces (overlays) the CP/M CCP but does not permit user programs to overlay MicroShell, as it must remain present to perform many of its functions. All of this action is invisible to the majority of CP/M programs which use CP/M's design entry points to accomplish their functions. Some programs, mostly older ones, "look around" inside of CP/M to perform some function. This is "not cricket" from a software compatibility and transportability aspect and these programs may not operate properly with MicroShell. See Appendix C for a list of known compatible and incompatible programs.

Since MicroShell overlays the CP/M CCP, it performs all of the functions that the CCP normally performs. The following commands will be executed by MicroShell just as the CP/M CCP would execute them, except as noted:

```
DIR [afn]              - Directory listing
ERA [afn]              - Erase file(s) (Note: No "*.*" warning)
REN newname=oldname    - Rename a file
TYPE filename          - Type a file
SAVE nn filename       - Save "nn" 256-byte pages of memory be-
                         ginning at 100 Hex in file "filename"
drivename:             - Change default drive
user NN                - Change user  area to area #NN
                         (Note: MicroShell allows the user to
                         access user areas 0-31.  The CP/M CCP
                         only allows the user to access 0-15
                         while 16-31 can only be accessed from
                         a user program call to CP/M.)
```

We will assume that the MicroShell user is familiar with these CP/M commands. The new CP/M user may wish to review these commands in the Digital Research CP/M documentation.

The CP/M CCP also provides the user with certain line editing functions to change a command line which has been entered in error prior to executing it (e.g. backspace, delete, control: u, x, r, e, etc.) These line editing features are supported by MicroShell. Again, the new CP/M user may wish to review these commands in the Digital Research documentation.

A final CP/M function which is also provided by MicroShell is control P (or p) which toggles the printer on or off.

With the exception of the extended user area access provided by MicroShell (0-31 vice 0-15), and the "era *.*" warning message, all CP/M CCP functions execute identically under MicroShell and provide the same error messages as the CP/M CCP on encountering an error condition.

2.3 Forming MicroShell Command Lines

The process of entering a command to MicroShell is identical to CP/M, i.e. the main command or program name is entered first, followed by any arguments the command requires. For example:

```
stat *.*
```

"stat" is the main command (program) to be executed and "*.*" is an argument to "stat" telling "stat" to display a list of all files in the current directory (actually the current disk drive and user number.) MicroShell accepts an identical command. In addition, however, MicroShell permits additional arguments to be given to perform special MicroShell functions. For example, if the user typed:

```
stat *.* >statout
```

MicroShell would send the normal output of the "stat *.*" command to the file "statout" instead of displaying it on the console. This is called output redirection and is one of the UNIX features brought to CP/M by MicroShell. It is discussed in detail in Section 2.5. MicroShell's other features, input redirection, pipes, multiple commands on one line, etc., are similarly invoked.

The maximum length of a command line to MicroShell is 84 characters.

A note is in order here about how MicroShell works. In the previous example, "stat *.* >statout", MicroShell recognizes the ">statout" portion of the command, strips it from the command, finds and loads the file "stat.com". "stat" never knows that ">statout" was typed. It sees the command just as if "stat *.*"

was typed from CP/M! The other special MicroShell commands are similarly stripped out of the command line by MicroShell before it is passed to the "main command".

White Space in Command Lines: In general, "white space" (spaces or tabs) may be used freely in commands to MicroShell. In particular, "white space" is optional before and after redirection symbols ("<", ">"), pipe symbols ("|") and semicolons. One place there cannot be "white space" is between a redirection symbol or pipe symbol and a "+" or "-" modifier. For example: "stat *.* >+ out", "stat *.*>+out" and "stat *.* >+out" are all legal command lines, but "stat *.* > + out" is not.

The other MicroShell special commands are covered in detail in the following sections and summarized in Appendix E.

2.3.1 Escaping MicroShell Special Characters:

The characters ">", "<", "^", "|", ";" and "\" have a special meaning to MicroShell when used anywhere in a command line. If it is desired that they be used in a command line for non-MicroShell functions (i.e. the user program needs them) their special meaning may be ignored by MicroShell in one of the following two ways:

1. Precede the special character with an "\", i.e.:

```
save 1 \>file
```

will create a filename ">file". To get a "\" to be passed through MicroShell, type two "\" 's, i.e.:

```
save 1 \\
```

will create a file named "\".

2. Enclose the entire argument containing the special characters in quotes ("), i.e.:

```
save 1 "><|"
```

will create a file named ><|. (Note that quotes (") cannot themselves be enclosed in quotes but must be escaped with a \.)

The characters "+" and "-" have a special meaning after ">", "<" or "|" and must be escaped with a "\" if the user really wants to have the "+" or "-" as part of the file or command name. It is best to stay away from the special characters; they're not common in file names anyway.

To round out the discussion of special MicroShell characters, all of the shell flags (Section 2.4), when used by themselves on a command line, are recognized by MicroShell as special commands and cannot be escaped.

2.3.2 Multiple Commands on a Line:

MicroShell will accept multiple commands on one line with each command separated by a ";". For example, the MicroShell command:

```
era *.bak;dir;stat
```

is equivalent to giving the following command sequence to CP/M:

```
A> era *.bak
A> dir
   ------------------------------------------
            CP/M displays directory
   ------------------------------------------
A> stat
   ------------------------------------------
          stat displays space remaining
   ------------------------------------------
A>
```

The only difference is that the MicroShell user can give all three commands at once and watch them sequentially execute! He doesn't have to wait for each command to complete before entering another command. The benefit of this feature, which comes from the UNIX operating system, is that we as human beings think in terms of logical processes which may involve numerous commands to accomplish. If we can type in all of the necessary commands to carry out a logical process, we are then freed from that level of detail and can concentrate on the results or on the next process. Thus the computer and its software has done what it does best - keep track of multiple and/or repetitive tasks while the user is free to think about logical processes. In Section 4 we will cover one higher step in this same philosophy: putting a number of individual commands into a file -- a shell file -- and executing them by merely typing the name of the file.

Limitations on Multiple Commands:

There are several restrictions on the use of the multiple command feature of MicroShell:

1. The maximum command line length of 84 characters cannot be exceeded. The total command (i.e. all characters entered up to the carriage return) is counted - not just each individual, semicolon-separated command.

2. Only 17 arguments are permitted for each semicolon-separated command. That is: a main command and 17 arguments to the main command.

3. A shell file name (Section 4) causes any remaining semicolon-separated commands to be ignored. For example, if "copy.sub" is a shell file containing a number of MicroShell commands, the command:

```
era *.bak;stat;copy file1 file2 file3;stat
```

will result in the final ";stat" being ignored by MicroShell.

4. Any commands on a line after the MicroShell eXit flag "-x" are ignored, since MicroShell relinquishes control to CP/M after the "-x" flag is executed.

5. Any redirection specified in one command is not carried over to the next semicolon-separated command. For example:

```
stat *.* >statout;dir
```

will cause only the "stat" output to be redirected to the file "statout", while the directory display from "dir" will come to the console as it normally would. This is not really a restriction; it is what would logically be expected to happen. It is mentioned to clarify the scope of action of redirection.

2.3.3 Interrupting a MicroShell Command:

If a series of commands are given on one line, typing ESCAPE or DELETE (RUBOUT on some keyboards) will interrupt MicroShell between semicolon-separated commands. This assumes that whatever program is running does not "gobble up" any input typed while it is running. As with interrupting CP/M's "submit", you may have to hit the ESCAPE key repeatedly. This interrupt feature also works to interrupt shell files.

If a particular program has its own mechanism for being interrupted (e.g. typing any key during a "type" or "dir" function) MicroShell will not inhibit its action. For example, if the following command line is typed:

```
dir;ed filename
```

and the user hits a key during the "dir" display, the "dir" process will terminate and MicroShell will execute "ed filename". Alternatively, if the user typed two ESCAPEs, the first would interrupt the "dir" process and the second would interrupt the whole command line. MicroShell would return with the prompt for the next command.

2.4 MicroShell Flags

Several of MicroShell's features can be altered by the user from within MicroShell itself. These commands to MicroShell are called flags.

Summary of MicroShell Flags

Flag	Meaning	Default*
F	File Search (On/Off)	On
G	Gobble line feeds (On/Off)	On
L	Login disks	-
P	Prompt change	%
S	Status report of flags	-
U	Upper case command line	On
V	Verbose mode: command echo	Off
X	eXit MicroShell to CP/M	-

* Default value may be user-customized with the CUSTOMIZ program.

2.4.1 Issuing a Flag Command:

A flag command is issued as a normal command to MicroShell, either on initial startup, or in response to the MicroShell prompt, or in a shell file or as one of multiple (semicolon-separated) commands on a line. The flag is preceded by a "+" or a "-" depending on whether the user wants to turn the feature on (+) or off (-). Flags may be issued in upper or lower case.

For example:

```
% -F<carriage return>
```

turns off the automatic file search feature while:

```
% +V<carriage return>
```

turns the Verbose mode on. If there is not an "On/Off" function associated with a flag, as with the "S" status report flag, either a "+" or a "-" may precede the flag. The following command line is legal:

```
% +V;-L;-U;-S<carriage return>
```

and would turn on the Verbose Mode, login all disks again, turn off Upper case command line translation and print a status report of the new flag conditions.

2.4.2 F flag: File Search (On/Off) Default: On

The "F" flag turns MicroShell's automatic file search feature on (+)(default) and off (-). The automatic search features are explained in Section 3. The F flag only affects file search not command search. If the user does not want MicroShell to search for files that match MicroShell-responsible extensions (.COM, .OVR, and .INT in the as-delivered MicroShell), the command:

```
% -F
```

will turn automatic file search off. It will remain off until the user either turns it back on with:

```
% +F
```

command or MicroShell is restarted from CP/M (if the user hasn't changed the default F flag condition with CUSTOMIZ.)

This flag is useful in cases where a program might be trying to erase or rename a file and the user doesn't want that to happen.

The automatic file search feature is really provided to permit a program to read files - like Wordstar reading its .OVR files or CBASIC loading .INT files, etc. When programs are erasing, rewriting or renaming files which have MicroShell-responsible file extensions, the user should be very careful about using the automatic file search feature. Be sure you know what's happening before files are erased or changed forever.

Two common programs fit exactly this category, where automatic file search is not desired: CP/M's PIP file copy utility and several available programs named COPY. So MicroShell explicitly recognizes when the user has invoked these two programs and turns the F flag off. When PIP or COPY is complete, MicroShell restores the F flag to its previous condition.

2.4.3 G flag: Gobble Line Feeds (On/Off) Default: On

During input redirection, MicroShell is often reading a file of commands into a program -- for example an editor script. When the file of commands is created with most editors, a carriage return followed by a line feed is placed at the end of each line.

Now comes the confusing part. Programs can ask CP/M for input characters from the keyboard in two ways; they can ask for

one character at a time (BDOS call 1 or 6) or a line of characters until a carriage return is typed (BDOS call 10.)

Now consider our file which we want to use as input. If the program calls for a line of input characters, MicroShell will supply successive lines from the file, automatically stripping the carriage return and line feed from the end of each line. Since the program called for a line of input, MicroShell knows that it shouldn't send the carriage return and linefeed. This is the most common call for input characters. DDT takes its input from this type of call, as does the command mode of the CP/M editor, ED. So everything works fine.

On the other hand, if the program calls for its input one character at a time, the linefeed after each carriage return in a file is probably going to confuse most programs. Leaving the linefeed out of the script file is one solution; but when the file is "type"d, all the lines appear on top of each other, distinctly confusing the user. Having MicroShell always throw away linefeeds after carriage returns is an alternative solution. This works fine for most programs, like CP/M's "ED" in the insert mode and most editors. But what if you wanted to write a program that took each character of a file, did something with it and passed it on to the output? This type of program is called a filter and is frequently used in the UNIX environment. If you want MicroShell to do the file handling for you, redirecting input and output so you can write a simple keyboard to screen filter, then you don't want MicroShell eating up the linefeeds!

So after all this explanation, MicroShell has a flag to selectively gobble linefeeds on single character input -- you choose. Note that MicroShell always gobbles a linefeed after a carriage return when a program asks for a line of input characters; you can't change that (and probably wouldn't want to.)

If you find you want this flag off most of the time, if you're using filters a lot instead of editor scripts, use CUSTOMIZ to change the default. Also remember that you can make up a shell file which sets this flag, executes some command and then restores the flag.

2.4.4 L flag: Login Disks

The L flag is provided to login a new disk, if disks are changed after MicroShell is started. If a disk is changed after MicroShell has accessed it once, then CP/M logs the disk as Read/Only and any attempt to write on the disk will result in a BDOS error. MicroShell exits to CP/M and warm starts CP/M after all BDOS errors.

Issuing the L flag (either +L or -L or +1 or -1, there's no difference), will cause MicroShell to login drive A, login the disk on the drive the user is currently on (the default drive)

and reset all other disks in the system. Then if the user selects a new drive, CP/M will log it in as read/write.

For example, if the user is currently on drive B and wants to put a new disk in drive B, he performs the following steps:

1. Change the disk in drive B.
2. Type -L or +L followed by a carriage return.

The head will load on drive A to login A and then on B to login B, (since B was the default drive.) Both drives A and B are now logged back into CP/M for read and write. If another drive is selected after the -L, CP/M will log it in again, since it "forgets" all disks logged in prior to the -L command.

All of this happens without the warm start that usually happens in CP/M. This is good and bad. Good because it's shorter. Good because on some double density systems it permits a single density disk without CP/M on the system tracks to be logged into drive A, permitting for example, PIPing between two single-density disks. Why bad? Bad because in some double density systems, density selection only occurs during a real warm start. On these systems, MicroShell must be exited to log in a different density disk. Try your system to see what it requires. The CompuPro system we use will properly determine disk density within MicroShell.

2.4.5 P flag: Prompt Change Default: "% "

The P flag permits the MicroShell prompt to be changed from within MicroShell. The following are examples of possible prompts:

```
%                              (default)

Drive: A User:0
%

%(bell)                        (bell sounds after %)

On Drive:A User:0              (Wordy!! but possible)
Enter command:
```

The flexibility for setting the prompt to match the user's needs is available. Short prompts for experienced users; long for occasional users; bells for users who do something else while their programs run!

This flag has an argument: the new prompt string. For example, the proper command to get the default prompt (back) is:

```
-p "%n%% "      (Note the space after the last "%")
```

It looks like a lot to type to just get a "% " to type!?

Well, here are the syntax rules for the prompt string. The syntax is a rough takeoff from the formatted print syntax for the C programming language.

Character	Meaning
%	Special character: next character means something special
N or n	New line: carriage return, line feed. %n means substitute a carriage return and linefeed. Note that unless you want a really unusual display, most prompt strings should begin with %n.
D	Upper case drive letter. %D substitutes the currently-selected drive in the prompt string.
d	Lower case drive letter. %d as above.
U or u	User number. %u substitutes currently selected user number into prompt string.

Since "%" has special meaning, to get a "%" to print in the prompt string, enter two "%"'s - %%. If embedded spaces or tabs are desired in the prompt, enclose the whole prompt string in quotes, like we did for the default prompt. (When using CUSTOMIZ to set the initial prompt string, quotes are neither required nor allowed when spaces are embedded in the prompt string.) If a bell or some other control character is desired in the prompt, just type it into the prompt string.

Now you can build your own prompt. You can put the prompt command in a shell file and execute it when desired by typing the name of the shell file. Look at our examples, "normprmp.sub" and "fullprmp.sub", on the distribution disk.

Maximum Prompt Length: 40 characters.

Note that with an intelligent terminal, some really unusual displays can be generated. For example, you could always have the prompt clear the screen, or change pages.

2.4.6 S flag: Status Report

The S flag provides a display of the status of the four flags which can be set on or off. The S flag can be issued as +S or -S (or +s or -s). The report looks like:

```
File search: On
Gobble lfs:  On
Ucase cmd:   On
Verbose:     Off
```

2.4.7 U flag: Upper Case Command Line (On/Off) Default: On

The U flag permits the command line passed to a program to either be in the CP/M-compatible upper case only mode or in upper/lower case as the user types it. (This is the command line which is set up at 80 Hex by the CP/M CCP or MicroShell prior to executing a program.)

Regardless of the state of this flag, MicroShell translates all file names (i.e. the "main command" name and the setup of the secondary file control blocks at 5C Hex) to upper case to remain compatible with CP/M upper case file names.

It is often desirable to pass a program upper and/or lower case arguments. Issuing the -U flag will permit this. Note, on the other hand, that some programs will not like to see lower case arguments, so use this capability with caution.

2.4.8 V flag: Verbose Mode: Command Echo(On/Off) Default: Off

The V flag permits the user to have MicroShell echo each command line just prior to executing it. This is useful when running shell files to see that argument substitution is occurring as intended. When executing a line with multiple commands in it (i.e. semicolon-separated commands) each individual command is echoed to the console prior to execution.

Redirection of output does not redirect the command echo if the V flag is on; the echo is always displayed on the console.

The Verbose flag only echoes command lines not all input to the shell. For example, if a shell file contains inline data to a program after a command line, only the command line is echoed to the console -- the data is not. This may take some getting used to for a CP/M user who is used to "submit" echoing everything in the submit file to the console. Likewise, if the input is redirected from a file, e.g. a file of commands to an editor, the input data is not echoed to the console.

2.4.9 X flag: eXit MicroShell

The X flag permits the user to exit MicroShell back to CP/M. A warm start of CP/M occurs leaving the user at the CP/M prompt.

See Section 4 on shell files for a method of exiting MicroShell to execute a CP/M command and then automatically reloading MicroShell. This is useful for running large programs which need the space used by MicroShell.

2.5 Output Redirection:

Output redirection is the process of redirecting a program's output from the console (i.e. the user's terminal) to a file. This capability is one of the UNIX features which MicroShell provides. For example, the CP/M command:

```
stat *.*
```

would normally send the CP/M utility "stat" output to the console. Under MicroShell, if the user types the command:

```
stat *.* >filename
```

MicroShell will redirect the output of "stat" to the file "filename". White space, blanks or tabs, are not permitted between the ">" redirection symbol and the filename. Using output redirection, the output of any program which runs under CP/M may be captured in a file for later use, editing, printing, etc. Creating documentation for a program is now much easier and more accurate since the exact screen output can be saved in a file for later incorporation into a manual. Or consider the problems of debugging a program which sends special characters to the CRT (e.g. cursor positioning, screen-clear, etc.). It's often hard to tell what's happening if the wrong special characters are being sent. With MicroShell, just redirect the output to a file and then look at the file with an editor or DDT to see what characters were sent.

2.5.1 Appending Output to a File:

To append output to an existing file, a modification of the basic output redirection function is also provided. If a user desires to place the output from a program at the end of an existing file, he types the following command line to MicroShell:

```
progname >>filename
```

MicroShell places the output of the program "progname" at the end of the file "filename". White space, blanks or tabs, are not permitted between the ">>" redirection symbol and the filename.

2.5.2 Echoing Redirection to the Console:

Both output redirection (">") and appending (">>") may be echoed to the console by following the output redirection symbol with a plus sign ("+"). Thus, the command:

```
stat *.* >+filename
```

would result in MicroShell placing the output from "stat" in the file "filename" with a simultaneous echo of the output to the console (i.e. a normal console display.) This permits the user to see what MicroShell is putting into the file and to respond to program prompts if the program requires inputs from the user.

2.5.3 Cautions in Redirecting Output:

1. If output containing control characters is redirected, some care must be exercised with control Z's (1A Hex). Since a control Z is used by CP/M and many CP/M programs to mark the end of a text file, the first occurrence of a control Z is often considered to be the end of the file. To most of the editors and word processors around, the rest of the file just isn't there. So if output containing control Z's is redirected to a file, a simple utility can be written as a filter to change the control Z's to an unused character (like ~). MicroShell's input redirection feature can be used to feed the file to this utility and the output redirected to another file, or a pipe can be used.

2. When redirecting a program's output to a file, the size of the file often grows faster than one would expect. MicroShell will issue an error message if the disk fills up while it is redirecting output:

```
Disk Full
```

3. If a program does direct output to the BIOS, bypassing the CP/M BDOS, MicroShell cannot redirect its output. Where possible, these programs should be modified to use the CP/M 2.2 direct I/O system call (BDOS function 6.)

4. Input that the user types into a program whose output is being redirected is displayed on the console but not redirected. See App. B for a method of invoking a second shell under MicroShell which will redirect all output (except direct BIOS calls.)

2.6 Input Redirection:

Input redirection is the process of redirecting a program's input from the console (i.e. keyboard) to a file. For example, the CP/M command:

```
ed filename
```

would normally require the user to enter various editing commands from the keyboard. If the user is making identical changes to a number of different files, entry of the editing commands from the keyboard is not only repetitious but also error-prone. Under MicroShell, the user can place the proper editing commands in a file, say "edscript", and cause "ed" to take its input from the file "edscript" by typing the command:

```
ed filename <edscript
```

Any program which requires keyboard input can therefore take its input from a file instead of the keyboard. As with output redirection, blanks and/or tabs are not permitted between the "<" redirection symbol and the filename.

2.6.1 Redirecting Console Status:

Since some programs make a console status call to CP/M

before calling for input data, MicroShell includes a provision for returning "character ready" to a program. If the user types:

```
progname <-inputfil
```

MicroShell sees the "-" as a command to return "character ready" on program requests for console status. This has been made a separate command in MicroShell because some programs sample "console status" as a means of determining if the user is attempting to interrupt or abort the program (e.g. "type", "dir", "ddt", etc.) In these cases the user would not want to redirect status with the minus sign "-".

2.6.2 Normal Termination of Input Redirection:

When redirecting input to a program, normal termination will occur when the program completes its action and exits back to CP/M. This exit will occur in one of three ways:

1. Program executes a return instruction. This is what programs do that are designed not to overlay the CP/M CCP. Without MicroShell, when they exit, CP/M prompts for the next command without doing a warm start. Under MicroShell, the MicroShell prompt occurs (after MicroShell closes any output redirection files.)

2. Program jumps to location 0 which is the warm start entry point for CP/M. Without MicroShell, CP/M normally does a warm start before prompting for the next command. Under MicroShell, action is the same as in #1 above; no warm start occurs.

3. Program calls CP/M "System Reset" BDOS call (0). Without MicroShell, CP/M executes a warm start as in #2 above. Under MicroShell, action is the same as in #1 and #2 above.

Input redirection does not alter the exit that a program makes but the user must remember that the input file must have the same characters in it that would be typed if there were no input redirection. I.e., if a program interprets a blank line (Return only) to mean exit, then the input file must end in a blank line. If the program expects a control C or control Z, then the input file must have that control character in it. These control characters make for awkward editing (especially control Z) and if there's any other way to exit the program, it's a lot better to use it and stay away from control characters. This may seem difficult to grasp at first, but experience will make it clear.

2.6.3 Input Redirection Errors:

There are two possible error messages which MicroShell may generate during input redirection:

End input file: input from console

If a program attempts to read past the physical end of an input file during input redirection from the file, MicroShell will issue this message and shift the input source to the console. Note that MicroShell does not stop at control Z's for two reasons:

1. To permit non-text files to be used in input redirection.
2. To permit control Z's in text files, e.g.: editor scripts where control Z's need to be read by the editor, and files with screen control characters in them.

If this error message occurs, in most cases there's a problem. The file probably has not caused the program reading it to terminate normally. But to keep things from blowing up, MicroShell gives the input back to the user in these cases.

Too many characters from input file

When input redirection is in progress for programs which are reading lines of buffered input from a file (BDOS call 10), the calling program establishes a maximum allowable buffer length into which the characters are placed by CP/M. Without input redirection, if the user attempts to enter more characters into the buffer than its maximum size allows, CP/M merely terminates the call for a line of input and returns to the user. This is occasionally seen in DDT when more than 31 characters are typed. During input redirection on buffered input calls, MicroShell monitors the maximum buffer size of the caller to ensure that the buffer is not overflowed. This situation is normally caused by some error that results in a line of input from an input file being too long for the caller's buffer. After sending the error message, MicroShell terminates the current command, any shell file or multiple commands that may be pending and prompts for the next command.

2.7 Pipes:

A powerful feature of the UNIX "shell" is the ability to "pipe" the output of one program to the input of another program. This permits building a powerful command from a series of simple tools. If the user types:

```
prog1 | prog2 | prog3 | ....
```

the output from "prog1" is sent to the input of "prog2" and the output of "prog2" is sent to the input of "prog3". The vertical bar ("|") is the command for MicroShell to create a pipeline. An up-arrow ("^") is equivalent to the "|" for MicroShell, since

some terminals don't have the vertical bar. "Pipes" are actually a shorthand for output redirection of the first program followed by input redirection of the second program. (The UNIX user will excuse this simplification of the full UNIX inter-process communication capability.) MicroShell permits the same output and input modifiers, "+" and "-", to be used with pipelines to achieve simultaneous console echo and/or console status redirection. For example, the command:

```
prog1 |+ prog2
```

would echo the output of "prog1" to the console as it was being fed to "prog2". The command:

```
prog1 |- prog2
```

would cause MicroShell to send "character ready" status to "prog2" for its input. These two modifiers may be combined in the same pipe as:

```
prog1 |+- prog2
```

The order of the "+" and "-" is not significant.

The MicroShell "pipe" function is implemented using temporary files which MicroShell erases after the "pipe" is complete. The MicroShell pipe function is better understood by knowing that internally MicroShell actually performs a pipe as follows:

```
prog1 >pype1;prog2 <pype1 >pype2;prog3 <pype2 ...
```

The names for the temporary pipe files -- "pype#" -- were chosen to avoid conflict with a user's existing file names. Our apologies to anyone who had been using "pype1", "pype2", etc. MicroShell will erase the temporary pipe files after the pipe is successfully completed (after the input redirection step.) If a pipe does not terminate normally due to a full disk or an input redirection error, the temporary file will be left on the disk for the user to view, erase, etc. Temporary pipe files are always written to the default disk drive (i.e. the drive from which the command was initiated.) The user must insure that sufficient space exists on the drive for a temporary file containing all the output that the program being piped will generate. If the user wants to use another disk drive for the temporary file, explicit output and input redirection is necessary, e.g.:

```
prog1 >a:temp;prog2 <a:temp;era a:temp
```

2.7.1 Error Messages from Pipes:

MicroShell has no special pipe error messages. Appropriate output and/or input error messages will be issued if appropriate.

3.0 Automatic Program Search

MicroShell contains two automatic search features to simplify operation under CP/M: main command (i.e. the "com" file) search and file search.

3.1 Main Command Search: If the user enters the command:

```
stat *.*
```

to CP/M, CP/M first looks for the program "stat.com", loads the program from the disk into main memory and then executes the program. The program to be executed will be called the "main command". CP/M will only "look" in one place for the program - on the current disk drive under the current user number. If the user is logged onto disk drive B in user number 0, CP/M will look on disk drive B for a file "stat.com" in user area 0. If CP/M does not find "stat.com" on the current drive in the current user number, CP/M will display the error message:

```
stat?
```

and prompt the user for another command.

MicroShell expands CP/M's normal search for a file to execute as follows:

1. MicroShell looks first on the current drive in the current user area for a shell file with the same root name, e.g. in the above example "stat.sub". If this file is found, MicroShell assumes the file contains text -- commands to be executed by MicroShell. See Section 4 for more information on shell files. If a shell file is not found, MicroShell begins looking for a ".com" file on the current drive and user number. If found, it is loaded and executed.

2. If the file is not found in step 1, MicroShell "looks" at the current user number. If the user number is greater than 0, MicroShell "looks" on the current disk drive in the user 0 area for the file. If found, the file is loaded, the user number restored from 0 to the initial user number and the program is executed. If the file is not found in user 0 (or user 0 was the initial user area), MicroShell's search proceeds to step 3.

3. MicroShell now looks at the search path specified by the user in the MicroShell CUSTOMIZ procedure (Section 5.) The search path contains from 0 to 3 additional disk drives for MicroShell to check. As delivered, disk drive A is specified in the search path, but the user may customize the search path to fit the needs of his system. MicroShell continues the search for the "main command" in user area 0 of the disk located on one of the specified disk drives. If the file is found, MicroShell loads it, restores the user number and default disk drive to their initial values and executes the program. If the "main command" is not found after searching all drives in the search

path, MicroShell issues an error message, e.g.:

stat? (or whatever the command name was)

and prompts the user for another command.
Example: The user is initially logged onto disk drive B in user number 1 and issues the command:

stat *.*

The search path will be assumed to be the as delivered path containing only drive A. MicroShell's search for "stat.com" proceeds as follows:

Drive	User
B	1
B	0
A	0

The user may override MicroShell's normal search path by specifying an explicit disk drive in the command. When an explicit disk drive is given, MicroShell looks first in the current user number on the disk drive specified and then in user 0 of the drive specified. For example, if the user issued the command:

a:stat *.*

from disk drive B, user area 0, MicroShell would immediately look on disk drive A, user 0 for "stat.com". If "stat.com" was not found there, MicroShell would not look further but would issue the error message:

a:stat?

and prompt the user for the next command. The user may want to use this feature if different versions of a program exist on two separate drives and the user specifically wants the version on another drive.

If the user is logged onto disk drive B, in user area 1 and issues the command:

a:stat *.*

MicroShell will look first on disk drive A in user area 1 for the file "stat.com". If the program is not found there, MicroShell will look in user area 0 on drive A before reporting failure.

3.2 Benefit of Main Command Search:

This feature may seem complex after reading the above discussion, but MicroShell's action is actually simple and logical; the user is freed from either prefacing commands with a disk drive or having to have a copy of all programs on every disk (and user area.) The concept of a "system disk" analogous to UNIX's "/bin" directory is not only feasible but greatly simplifies the whole operation under CP/M. If the user keeps a

disk with all routinely used programs in disk drive A, he can then operate from drive B, in any user area, without concerning himself with where his programs are located. The user merely types the desired command line and MicroShell does the work of finding the program.

For single drive systems, the user can move to any user number on drive A without copying his programs to that user number.

Users with a hard disk may wish to "customize" MicroShell to reflect their system configuration using MicroShell's CUSTOMIZ program (Section 5). If the hard disk is drive C, for example, the user may wish to change the as delivered search path from drive A to drive C. Or, using CUSTOMIZ, the user could specify that drive C be checked first (after the default drive check which cannot be omitted) and then drive A. The flexibility is available to satisfy the user's needs.

3.3 File Search:

As a separate search function, MicroShell performs a search for any files which a "main command" i.e. a program, may attempt to access. If, for example, the user issued the command:

```
ddt file.ext
```

from drive B under CP/M, (without MicroShell), CP/M would load "ddt.com" from drive B and execute it. DDT would then attempt to load the file "file.ext" from drive B. If DDT didn't find "file.ext" on drive B, it would issue the error message "?" and await another command.

MicroShell "oversees" all attempts of a program to open a file. If the attempt by the program to open a particular file is unsuccessful, MicroShell "steps in" to help out the program as follows:

1. MicroShell has a list of file extensions or file types for which it is responsible. The file extension is ".ext" in the example above; in general, the file extension is the part of a filename after the ".". If MicroShell sees that a program was unsuccessful in opening a file, and the file extension of the file name is one for which MicroShell is responsible, an automatic file search is performed.

2. MicroShell uses the same search path that it uses for "main command" searching to look for the file which the user program is trying to open. Since MicroShell knows that the program was already unsuccessful in opening the file from the current disk drive and user number, it is not rechecked by MicroShell. Instead, user number 0 of the current drive is checked first if the program was running in a user number greater than 0. If the file is found in user 0, MicroShell opens it and then returns control to the program which can then read and/or write

from the file just as if the file was in the current drive and user area.

3. If MicroShell does not find the file in user area 0 on the current drive, the search path specified for "main command" search is used to attempt to find the file. I.e., with the as delivered search path of drive "A", MicroShell would look on drive A, user area 0 for the file. If found, MicroShell would open the file and return control to the user program to use the file. (Note: MicroShell changes the disk drive byte in the file control block to reflect the drive the file was found on.)

4. If after looking down the search path for the file, MicroShell cannot locate it, MicroShell returns control to the user program with the normal CP/M open failure code. The user program will then proceed to do whatever it does when it can't find the file.

3.4 Benefit of Automatic File Search:

The benefits which MicroShell provides with the file search feature are similar to those of the "main command" search; the user does not need to be concerned with where all the files that a program needs are located. He merely executes the programs as if all the necessary files were on the current disk drive and user area and MicroShell handles the tedium of locating the files.

3.5 Some Practical Applications:

1. One popular word processing program, Wordstar, needs to access two overlay (".ovr") files for normal operation. Though Wordstar permits one disk drive other than the current drive to be searched for these files, no check across user areas is supported. Under MicroShell, the overlay files may be on the system disk, and MicroShell will find them for Wordstar from any user number or disk drive. See Section 3.6.7 below.

2. Many Cbasic application programs "chain" to subsequent ".int" files from a main menu program. MicroShell can supervise this chaining process and permit all ".int" files to be kept on the system disk.

3. The C compiler used to develop part of MicroShell, BDS C, uses two ".com" files to compile a program. After completing the first pass, the first ".com" file, "cc1.com", chains to the second ".com" file, "cc2.com", to generate the compiled code. Although a command flag for "cc1" is available to specify the drive from which "cc2" is to be loaded, MicroShell can relieve the user of this detail.

3.6 Some Practical Limitations:

There are some limitations which the user should keep in mind in using MicroShell's file search features.

3.6.1 MicroShell permits up to three disk drives to be automatically searched. The disk drive search path is set by the user with the CUSTOMIZ program. As delivered, MicroShell has only Drive A in its search path. The user should remember that each drive which MicroShell checks requires a finite amount of time: the drive must be selected and the directory read by CP/M. This time will depend on the user's particular computer system and disk drives and how many drives are checked. Therefore, the user should make the search path as short as possible while satisfying his particular needs.

3.6.2 To accomplish the automatic file search, MicroShell must compare the file extension on every failed attempt to open a file, with the list of file extensions for which MicroShell is responsible. Six extensions are permitted to be MicroShell-responsible file extensions. In the as delivered program, these are set to ".com", ".ovr", and ".int" but the user may change these using the "install" program. As with the search path, the user should specify only those file extensions which are necessary for his situations.

3.6.3 There will be times when the user may not want MicroShell to do any automatic file searching. MicroShell has a flag ("-f" - Section 2.4.2) which permits the file search feature to be turned off and on from within MicroShell. Thus the user can turn it off while he runs one particular program and then turn it back on. Or he may create a shell file (Section 4) to take care of the details of this flag and to execute a particular command. The status flag ("-s") may be used to determine if file search is turned off or on.

3.6.4 Some programs scan the disk directory for a desired file prior to attempting to open it. If they don't "see" the desired program in the directory, they don't try to open the file. MicroShell can't help these programs because it is not possible to easily determine what file the program may be searching for (e.g. it may be using a wild character, "*" or "?", in the search.)

3.6.5 Some programs look for a file to erase before or after performing some action. Copy utilities (like CP/M's PIP) are an example of these. The user must remember that if file search is enabled and the file extension of the file being erased matches one in MicroShell's list, it will look down the search path for the file. This can have some undesirable effects if the user is not aware of what is happening. (like the wrong file gets erased!) Be careful in selecting the MicroShell-responsible extensions. A shell file (Section 4) which turns off the file search feature, executes the desired program and then turns the file search back on is one way to solve this problem. Two specific programs are used frequently enough to warrant MicroShell automatically handling this situation: PIP and COPY. (COPY was chosen as the frequent name given to many copy programs.) MicroShell will automatically turn off the file search flag when

either of these programs are run and then turn it back on (if it was initially on) after the programs terminate without any user action. If for some reason the user really wants the file search feature to be active with these features, the programs can be renamed so that MicroShell will not recognize them.

3.6.6 Automatic Searches and Shell Files: After a user issues a command line to MicroShell, the first thing MicroShell looks for is a shell file to execute. For example, if the user types:

```
stat *.*
```

MicroShell first looks on the current drive (in the current user area) for a file named "stat.sub" (or whatever the shell file extension may have been customized to by the user.) It is important to note that MicroShell only looks on the current drive and in the current user number for a shell file; it does not do an automatic search for shell files. If it finds "stat.sub", MicroShell assumes that it is a file of commands. The file is read and the commands in it are sequentially executed by MicroShell. If a file "stat.sub" is not found, MicroShell begins a search for "stat.com", looking down the search path described previously if necessary. So, in order to execute a program (a "com" file) MicroShell makes at least two disk accesses: first to look for a shell file by the same root name and then for the "com" file. Automatic searching for shell files was not incorporated because of the increased time it would have caused in normal program location and loading.

Note that shell files need not reside on the default disk drive. If the shell file "compile.sub" is on drive A and the user is logged into drive B, typing "a:compile" will locate and execute the shell file.

3.6.7 Automatic Searches and Wordstar: MicroShell's automatic file search feature operates properly with the Wordstar word processing program to locate "ws.com" and the necessary ".ovr" overlay files. Two points need to be observed. First, do not set any attributes on the files, i.e. do not make them "system" or "read-only" files. Second, Wordstar has its own built-in capability to check another drive for the overlay files. This feature can conflict with MicroShell's automatic file search facility if it is not set up properly. The Wordstar variable is "DEFDSK:" and can be set using the Wordstar install program. See page 8-6 of the Wordstar Version 3.0 manual. If "DEFDSK:" is set to "0", Wordstar will do no searching itself and MicroShell will handle finding the overlay files. Alternatively, if the overlay files will always be on the same drive, it may be set to the number for that drive (A=1, B=2, etc.) The latter method permits proper operation if Wordstar is used outside of MicroShell but requires that the overlay files always be on the drive specified. With "DEFDSK:" set properly, Wordstar may be invoked from MicroShell in any user area on any disk and MicroShell will locate "ws.com" and the overlay files automatically.

4.0 Shell Files

MicroShell's shell file capability provides all of the functions of CP/M's "submit" feature and adds some additional features. This capability allows the user to have MicroShell execute a series of commands from a file. This file is often called either a shell file or a command file.

4.1 Constructing the Shell File:

Using an editor, the user types the desired command lines into a file. ("pip filename=con:" may be used to more quickly create a short file. Remember to explicitly type the line feed after a carriage return which the editor normally adds automatically.)

The filename for the shell file should have an extension of ".sub" to allow MicroShell to recognize it as a shell file. This extension may be changed by the user by the CUSTOMIZ program. (See Section 5.)

Any of MicroShell's features may be included in the commands placed in the shell file with the following restrictions:

-X flag:	This is the last command MicroShell will execute before exiting to CP/M.
Another shell file:	Another shell file may only be placed as the last command in a shell file since MicroShell will not execute nested shell files.

With these restrictions, all other MicroShell commands including redirection, pipes, multiple commands on a line, shell flags, and CP/M commands (ERA, DIR, REN, TYPE, USER, default drive selection) may be included. MicroShell will perform normal disk drive searches for programs specified in shell files as described in Section 3.

4.2 Executing Shell Files and Argument Substitution:

To execute a shell file under MicroShell, the user types the name of the shell file, with or without the extension, followed by any arguments to be substituted in the commands in the shell file.

For example, if the file "show.sub" contained the following line:

```
type $1;stat $1
```

and was executed by typing:

```
"show doc"
```

MicroShell will substitute "doc" for all occurrences of "$1" in the shell file and execute the commands. The user can see the command lines after substitution has taken place, prior to MicroShell actually executing them, by turning on the "verbose" (+V) shell flag.

MicroShell accepts a total of 18 arguments, including the argument containing the shell file name ("show" in the example above.) MicroShell will in fact substitute the shell file name as typed for any occurrences of "$0" in the shell file.

If the character following a "$" in a shell file is not numeric, MicroShell will recognize that it is not an argument substitution. If there are two "$"s in a shell file, MicroShell will substitute one "$" for the two, to remain compatible with the CP/M "submit" convention.

4.3 Null Arguments:

To permit shell files to handle a varying number of arguments, no error is generated for a missing argument and MicroShell merely "closes up" the command line with the "$n" for the argument that was not specified.

For example, the Digital Research Macro Assembler, MAC, will take an optional argument specifying certain options, i.e.:

```
MAC filename $PZ
```

The "$PZ" is optional. If the shell file "domac.sub" contained the line:

```
mac $1 $2;load $1;era $1.hex
```

and was executed by typing:

```
domac test $PZ
```

then MicroShell would issue to CP/M the commands:

```
MAC TEST $PZ
LOAD TEST
ERA TEST.HEX
```

If the second argument was omitted by the user, e.g.:

```
domac test
```

MicroShell would issue the commands

```
MAC TEST
LOAD TEST
ERA TEST.HEX
```

The space containing the "$2" in "domac.sub" would be "closed up" by MicroShell.

4.4 Control Characters in a Shell File:

If control characters are necessary in a shell file, they may be entered with an editor that will insert actual control codes, or they may be entered by the sequence "^C", where "C" is the letter associated with the control code (i.e. TAB = ^I). When MicroShell reads the line, it will substitute the appropriate control character for the ^C sequence. If a "^" is really desired in the file, it must be "escaped" with a "\" (i.e. "\^"). This substitution occurs when lines of a shell file are read and during input redirection when lines of buffered input (CP/M BDOS call 10) are read. It does not occur during input redirection when single characters (CP/M BDOS calls 1 and 6) are read.

4.5 Comments in Shell Files:

If a command line in a shell file begins with ":", the entire line is ignored by MicroShell in executing the shell file. This permits commenting of shell files for future edification, understanding, etc. Command lines may be "commented out" therefore by merely placing a ":" at the beginning of the line.

4.6 Input Redirection in Shell Files:

In addition to permitting normal input redirection, ("<") in a shell file command line, MicroShell provides a default redirection of input to the shell file if no other redirection of input is specified. This is analogous to the action of CP/M's "XSUB" placed in a submit file.

For example, if the shell file "disasm.sub" contained the following lines:

```
ddt $1.com >$1.asm
L100 $2
G0
```

and was executed by typing:

```
disasm test 1000
```

then the first line would cause MicroShell to load "DDT" with the file "test.com" loaded by "DDT" and output redirected to "TEST.ASM". When "DDT" called for the first line of console input, MicroShell would supply the line "L100 1000". DDT would then output a listing of "test.com" from 100 hex to 1000 hex which MicroShell would redirect to the file "TEST.ASM". When DDT called for a second line of input, MicroShell would supply the

line "G0" causing DDT to exit. MicroShell would then close the file "TEST.ASM" and return to the user with the prompt for the next MicroShell command.

This example demonstrates the power of MicroShell to make a more powerful tool by combining existing tools. The user could edit the resulting file, as desired, and reassemble it. A disassembler for free!

4.6.1 Changing the Default Input Redirection:

Input redirection to a shell file may be changed from the default input redirection from the shell file itself (which has the same effect as "XSUB" in a "submit" file) by issuing an explicit input redirection command in the shell file. In the above example, if the first line of the shell file "disasm.sub" were changed to:

```
ddt $1.com >$1.asm <script
```

then MicroShell would supply command lines to DDT from the file "script" instead of from the shell file "disasm.sub" itself. After the command with explicit input redirection terminates, MicroShell will revert back to redirecting input from the shell file on successive command lines, unless they also contain explicit input redirection commands.

The user should note that if DDT exhausts the command lines in "script" MicroShell will not shift the input back to the shell file. Some care must be used in writing files for input redirection to ensure they contain sufficient commands to cause their caller to eventually terminate. If a caller requests more input from a file than the file contains, MicroShell will issue the message:

```
End input file: input from console
```

and shift the input to the console for further input. As discussed in Section 2.6, this may result in garbage being fed to the program after the final control Z is sent (i.e. the data between the control Z end-of-file marker and the physical end of the file. Not all editors fill the final sector with control Z's after the end-of-file control Z.)

4.6.2 Redirecting Shell File Input to the Console:

If the user desires to be able to enter input to a program executed by a shell file from the console (keyboard), a special input file name "$T" is provided. Thus, in a shell file (and only in a shell file) if a command line contained:

```
ddt $1 <$T
```

DDT calls for input would be supplied from the console (keyboard). The next line which MicroShell reads from the shell file will undergo the normal input redirection process; i.e. if no explicit input direction is given, input is redirected to the shell file itself.

4.7 Interrupting Shell Files:

Either the ESCAPE key or the DELETE key (RUBOUT on some keyboards) will interrupt a shell file which is in progress. If the program currently running polls console status to see if an interrupt key has been struck, two ESCAPEs (or DELETEs) may be necessary to ensure one of them gets past the program and is seen by MicroShell.

4.8 Shell Files Which Return to CP/M:

It may be desirable to execute a command from MicroShell which requires all of the available working memory in the computer, overlaying MicroShell. This can be easily accomplished by creating a shell file which in turn creates a CP/M "submit" file and exits to CP/M to execute the "submit" file. The time overhead is not significant compared to the run time of programs which require that much memory. Here's how to do it:

Create a shell file, say "dolong.sub", which has the following lines in it:

```
a:
echo $1 $2 $3 $4 $5 >long.sub  (as many args as req'd)
echo a:sh b: >>long.sub
submit long
-x
```

This shell file will build a submit file, "long.sub", which "submit" will process to a "$$$.sub" file on drive A. "-x" then exits MicroShell to permit the CP/M CCP to execute the "$$$.sub" file.

To execute the shell file to do, for example, a compile with a compiler that requires all of memory, issue the command:

```
% dolong compile progname arg1 arg2 arg3 arg4

  [ MicroShell builds "submit" file and exits to CP/M ]
A>compile progname arg1 arg2 arg3 arg4

  [ CP/M CCP executes "$$$.sub" file to do compile ]
A>a:sh b:

  [ CP/M CCP reloads "sh.com" as last command ]
%
  [ MicroShell restarts, goes to drive B (or wherever
    you want, and waits for the next command. ]
```

Another example of a useful shell file which returns to CP/M is the following file, called "reboot.sub":

```
a:
echo a:sh b: >boot.sub
submit boot
-x
```

Execution of "reboot" will cause CP/M to warm start, reload "sh.com" and change to drive B. This may be used to login a different density disk on those systems requiring a warm start to change disk density. (For some systems, the "-L" login flag -- Section 2.4.4 -- will work. It depends on how the BIOS was written for a particular system.

5.0 MicroShell Customization

MicroShell, as delivered on the distribution disk, is set up for the "average" user but since many different environments are possible under CP/M, many users will want to "customize" MicroShell for their particular needs.

The CUSTOMIZ program distributed with MicroShell will permit the user to change several of MicroShell's features, including:

Sect	Feature	Default
5.2	Search Path	Drive A
5.3	MicroShell-Responsible File Extensions	COM OVR INT
5.4	Initial Prompt String	% (actual string="%n%% ")
5.5	Shell File Extension	SUB
5.6	Flag settings:	
	F - File Search	On
	G - Gobble Line Feeds	On
	U - Upper Case Command Line	On
	V - Verbose Mode	Off

5.1 Executing the CUSTOMIZ Program:

Running the CUSTOMIZ program is mostly self-explanatory. The program is executed by typing:

```
CUSTOMIZ
```

from either CP/M or MicroShell. It will sign on with:

```
MicroShell Customization Program - Version X.X
Copyright (c) 1981 New Generation Systems, Inc.
```

It then looks on the current drive for a copy of 'sh.com' to customize. If it does not find 'sh.com', it will ask for the drive that 'sh.com' is on. (Note: if CUSTOMIZ is executed under MicroShell, automatic file search may relieve you of this choice.) After locating and reading in 'sh.com', it asks where to write the new 'sh.com':

```
Enter drive to write new 'sh.com' on.
Note: If same drive as old 'sh.com', old 'sh.com' will be renamed
'sh.old'.
Drive:
```

It then waits for the user to enter the drive. With this housekeeping out of the way, CUSTOMIZ displays the current status of the user-changeable features:

```
**************************************************************
*                    Current MicroShell Setup                *
**************************************************************
1 - Search Path: A
2 - MicroShell-Responsible File Extensions: COM OVR INT
3 - Initial Prompt String: %n%%
4 - Shell File Extension: SH
5 - Flag settings:   F - File Search: On
                     G - Gobble Line Feeds: On
                     U - Upper Case Command Line: On
                     V - Verbose Mode: Echo Command Lines: Off
**************************************************************

Enter number of item to change (0 = no more changes):
```

The user then enters the number of the item he wants to change, 1 - 5. CUSTOMIZ returns to the main status display above after each item is changed to display the new values and ask if more changes are desired.

5.2 Selection 1 - Changing the Search Path:

To alter the Search Path, the user enters 1 and CUSTOMIZ displays:

```
**************************************************************
*                     MicroShell Search Path                 *
**************************************************************

Additional disk drives MicroShell searches after failing to find a
program on the current drive.  Maximum of 3.

Current search path: A

First  drive to check --> Current Value: A
                      Enter drive letter,
(Return=no change,0=end of search path):
```

The user is now given the opportunity to change the first drive that MicroShell checks (after the default drive) from the as delivered value of "A" to his choice. Since this is the first place MicroShell checks after the default drive, it should be the most likely place to find the majority of programs. For two drive systems, it should probably remain "A", leaving "B" as the "work disk". For hard disk systems, it should be one of the hard disk drives. For single drive systems, the user should enter "0" so that MicroShell does not waste time rechecking drive A.

CUSTOMIZ will now prompt for the second and third drives to check.

```
Second drive to check --> Current Value: None
                      Enter drive letter,
(Return=no change,0=end of search path):
Third  drive to check --> Current Value: None
                      Enter drive letter,
(Return=no change,0=end of search path):
```

For two drive systems, "0" should be entered for Second drive and CUSTOMIZ will return to the main status display. Hard disk system users with multiple drives logically located on the hard disk should enter the drive letters as the Second and Third drives for MicroShell to check. The Search Path feature is described in Section 3.1.

CUSTOMIZ then returns to the main status display and asks for the next item to change:

```
Enter number of item to change (0 = no more changes):
```

5.3 Selection 2 - Changing the MicroShell-Responsible File Extensions:

If the user types 2, CUSTOMIZ displays:

```
*************************************************************
*            MicroShell-Responsible File Extensions         *
*************************************************************

File extensions for which MicroShell will perform automatic file
search.

Current extensions: COM OVR INT
Enter/change extensions.  Maximum number is 6.

      Extension #1 Current Value: COM
                 Enter new extension
(Return for no change)(0 to end):
```

These are the file extensions for which MicroShell will perform automatic file search. Section 3.3 describes the Automatic File Search feature. The user may leave COM as the first extension for which MicroShell is responsible by typing Return, or may change the extension as desired (upper or lower case), or may end the MicroShell-Responsible File Extensions by entering "0". If "0" is entered, any existing extensions following #1 are deleted and CUSTOMIZ returns to the main status display. Otherwise, extension #2 is displayed:

```
      Extension #2 Current Value: OVR
                 Enter new extension
(Return for no change)(0 to end):
```

The user has the same choices here. #3 - #6 follow similarly:

```
      Extension #3 Current Value: INT
            Enter new extension
(Return for no change)(0 to end):

      Extension #4 Current Value: None
            Enter new extension
(Return for no change)(0 to end):

      Extension #5 Current Value: None
            Enter new extension
(Return for no change)(0 to end):

      Extension #6 Current Value: None
            Enter new extension
(Return for no change)(0 to end):
```

"None" means there is currently no extension entered for that extension #. Some discretion should be used in entering a lot of extensions at first until the user is familiar with the action of the MicroShell automatic file search feature.

After either "0" is entered to end the file search extensions or all six extensions have been entered, CUSTOMIZ returns to the main status display and prompts for the next item to change:

```
Enter number of item to change (0 = no more changes):
```

5.4 Selection 3 - Changing the Initial Prompt String:

If the user now types 3, CUSTOMIZ displays:

```
**************************************************************
*                Initial MicroShell Prompt String            *
**************************************************************

Initial prompt string.  See the MicroShell Manual for the proper
format.  A maximum of 40 characters are permitted in the prompt
string.
                  Current prompt: %n%%
Enter new initial prompt string
        (Return for no change):
```

This allows the user to enter any allowable prompt as the initial MicroShell prompt instead of the "% ". Section 2.4.5 of the manual describes the prompt string format. Changing this prompt does not affect the ability to temporarily change the prompt from within MicroShell.

After entering the initial prompt string, CUSTOMIZ returns to the main status display and prompts for the next item to change:

```
Enter number of item to change (0 = no more changes):
```

5.5 Selection 4 - Changing the Shell File Extension:

If the user now types 4, CUSTOMIZ displays:

```
**************************************************************
*                      Shell File Extension                  *
**************************************************************

MicroShell will directly execute files of commands with this file
extension.

                    Current shell extension: SUB
Enter shell file extension (3 characters)
                     (Return for no change):
```

The extension of the files which MicroShell recognizes as shell files (similar to CP/M's "submit" files) may be changed. The initial value is "SUB" to be compatible with CP/M "submit" files. If this extension is changed, the names of the demonstration shell files provided on the distribution disk should be changed accordingly.

CUSTOMIZ then returns to the main status display and asks for the next item to change.

```
Enter number of item to change (0 = no more changes):
```

5.6 Selection 5 - Changing the Initial Flag Defaults:

If the user enters 5, the initial flag default values can be changed. CUSTOMIZ displays:

```
**************************************************************
*                    Initial Flag Defaults                   *
**************************************************************
```

The default value of 4 of MicroShell's flags may be set by the user. See the MicroShell Manual for the meaning of each flag.

```
                    F - File Search: On
Enter desired default value (On/Off)
             (Return for no change):

                G - Gobble Line Feeds: On
Enter desired default value (On/Off)
             (Return for no change):

          U - Upper Case Command Line: On
Enter desired default value (On/Off)
             (Return for no change):

V - Verbose Mode: Echo Command Lines: Off
Enter desired default value  (On/Off)
              (Return for no change):
```

Each of the MicroShell flags is presented for the user to change the initial default value. Section 2.4 describes the function of each of the shell flags. Changing the initial default value does not affect the ability to change each of the flags from within MicroShell. CUSTOMIZ then returns again to the main status display and prompts for further changes:

```
Enter number of item to change (0 = no more changes):
```

5.7 Selection 0 - Ending the CUSTOMIZ Function:

If the user types "0", indicating no further changes are to be made, CUSTOMIZ writes the new 'sh.com' file to the specified disk and returns to CP/M (or MicroShell, if it was executed from MicroShell.) If there was already a "sh.com" file on the disk selected as the destination disk, it is renamed to "sh.old".

5.8 Setting Up CP/M for Automatic Loading of MicroShell:

If it is desired that MicroShell be loaded on initial startup of CP/M, the CP/M Console Command Processor (CCP) may be patched to accomplish this action. You will have to have a familiarity with the "SYSGEN" and "DDT" CP/M utilities to do this.

On initial entry, CP/M looks to see if there is a command present in the CCP buffer at the beginning of the CCP. If there is, the command is immediately executed without waiting for user

input. In a standard CP/M system, there are 127 bytes available in this buffer for the initial command.

Note: Some manufacturers have used part or all of this buffer for various purposes in their implementation of CP/M. If the buffer does not initially contain spaces and the Digital Research copyright notice, it may already be used in your configuration. Proceed with caution; if you patch over existing code and/or data, there may be a problem.

Follow the following steps to patch the CCP command buffer:

1. Obtain a memory image of your CP/M system. "SYSGEN" (or your system's equivalent function) is used to get the CP/M system from the system tracks of the disk in drive A. The following commands will do this in the "standard" CP/M system:

```
A>ddt
DDT VERS 2.2          (<-- DDT signs on and prompts:)
-f100 4000 0          (<-- fill memory with 0's)
-g0                   (<-- go back to CP/M)
A>sysgen
(SYSGEN signs on and prompts:)
Source drive name (or RETURN to skip).a (<-- answer "a")
Source on A then type return.           (<-- type RETURN)
Function complete.                      (<-- got system)
Destination drive name (or RETURN to terminate). (<-- enter
                                            a RETURN to this)
A>save 48 cpmsh.com (<-- this may save more than may be
                         necessary for your system but it
                         won't hurt.)
```

2. Load the memory image with "ddt" and find the CCP. It may be located at different memory addresses depending on how your system implementer configured CP/M. First, look at 980 Hex for it. That's the "standard" location.

```
A>ddt cpmsh.com
DDT VERS 2.2              (<-- DDT signs on and prompts:)
NEXT  PC
3100 0100
-d980                     (<-- this displays memory at 980 hex.)

                        -- Maximum length of command buffer
                        |
                        |  -- Length of current command in buffer
                        |  |
                        |  |  -- Start of command buffer
                        |  |  |
0980 C3 5C DD C3 58 DD 7F 00 20 20 20 20 20 20 20 20 .\..X...
0990 20 20 20 20 20 20 20 20 43 4F 50 59 52 49 47 48         COPYRIGH
09A0 54 20 28 43 29 20 31 39 37 39 2C 20 44 49 47 49 T (C) 1979, DIGI
09B0 54 41 4C 20 52 45 53 45 41 52 43 48 20 20 00 00 TAL RESEARCH  ..
09C0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
---------- (Remainder of "d" display omitted) ----------------------
```

(Note: Your display should look something like the output above with the addresses at 981-2 and 984-5 possibly offset depending on the size of your CP/M system. As an example of the power of output redirection, the display above was "captured" from "ddt" with the output redirection of MicroShell to avoid a lengthy and error-prone retyping of the data.)

3. If you don't see a similar display, go to step 4 below. If it looks the same (except for the addresses), you can enter your initial command as follows:

a. Decide what you want MicroShell to do on startup. You can just enter "sh" and MicroShell will load and display its initial prompt. Or you might want to run a shell file to display the directory, show the remaining size, etc. To run a shell file on initial startup, you could enter "sh startup" if the shell file name is "startup.sub".

b. Translate your desired command to upper case ASCII hex; "sh" would be 53 48. "sh startup" would be 53 48 20 53 54 41 52 54 55 50.

c. Enter the command into the buffer using the DDT "set" command, starting at 988 Hex, with the length of the command:

```
-s987           (<-- length of command goes at 987 Hex.)
987 00 2        (<-- Enter the total number of bytes
                     including spaces in your command.)
988 20 53       (<-- "S")
989 20 48       (<-- "H")
98A 20 0        (<-- 0 must terminate the command string)
98A 20 .        (<-- enter "." when done entering
                     command.)

-d980           (<-- check your command by displaying.)

0980 C3 5C DD C3 58 DD 7F 02 53 48 00 20 20 20 20 20 .\..X...SH.
0990 20 20 20 20 20 20 20 20 43 4F 50 59 52 49 47 48         COPYRIGH
09A0 54 20 28 43 29 20 31 39 37 39 2C 20 44 49 47 49 T (C) 1979, DIGI
09B0 54 41 4C 20 52 45 53 45 41 52 43 48 20 20 00 00 TAL RESEARCH  ..
09C0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
---------- (Remainder of "d" display omitted) ----------------------
```

4. If you didn't find the beginning of the CCP at 980 Hex, you have a non-standard CP/M configuration. Look through memory using the DDT "d" display command until you find the Digital Research copyright message. If you don't find it, chances are your system integrator has used this buffer space for his own purposes and it is not available to you. If you do find it, enter your command as in 3 above, adjusting the memory addresses you "set" accordingly. (To help you with one non-standard system, the Godbout CompuPro system has the CCP starting at 1600 hex in

the memory image.)

5. Exit DDT and SAVE the modified CP/M memory image.

```
-g0                          (<-- exit DDT)
A>save 48 cpmsh.com          (<-- overwrites old "cpmsh.com")
```

6. After copying MicroShell (and any files required by your startup command) to a new disk, SYSGEN the new disk with the patched CP/M that you saved in step 5 above.

7. With the new disk in drive A, MicroShell will load on startup (reset) and either prompt for a command or execute an initial command if you specified that. Note that the "-x" MicroShell flag will now cause an exit to CP/M, a warm start of CP/M and a reload of MicroShell. The only way to "escape" to CP/M with this patch installed is to erase or rename "sh.com". Then when MicroShell is exited with the "-x" flag, CP/M won't be able to find "sh.com", will display "sh?" and then prompt you for another CP/M command. Also notice that any time MicroShell gets a CP/M error (bad sector, read only, disk select, etc.), it will exit to CP/M and do a warm start. With the CCP patched to auto-load MicroShell, control will immediately be returned to MicroShell. If you get lots of errors (new user, etc.), this can be useful to avoid having to retype "sh" after CP/M errors.

APPENDIX A

History and Design of MicroShell

We began using CP/M in January 1978. It was a vast improvement in microcomputer operating systems; relatively easy to use, efficient in memory and disk usage and a tremendous bargain for its price. We happily used CP/M and its facilities to develop software adapting to its various features and limitations. Then in 1980 a revolutionary event occurred: we were introduced to the UNIXtm operating system developed by Bell Laboratories for the Digital Equipment Corporation PDP-11 minicomputer series. We were elated. Always having been shy about "big" computer operating systems and their complexity, UNIX pleasantly surprised us. It was easy to learn, easy to use and very powerful.

The user-interface to UNIX is its "shell" program, equivalent to CP/M's Console Command Processor (CCP). The idea for MicroShell really began when we used a DEC VAX minicomputer. Instead of being greeted at the terminal with DEC's operating system, VMS, we saw what looked like UNIX's "shell"! The Software Tools, which began in a book by the same name by Kernighan and Plauger and were later expanded by Lawrence Berkeley Laboratory at the University of California, had been installed "on top of" DEC's operating system. The Software Tools include a UNIX-like "shell" and give the appearance of running UNIX <u>without</u> losing compatibility with the native operating system or requiring the development of a whole new operating system for the VAX.

The idea was born! Why not develop a "shell" to lie "on top of" CP/M?! And in the spring of 1981, MicroShell was thus conceived.

<u>MicroShell Design</u>:

It was decided to implement the best functional features of the UNIX "shell" in MicroShell. The initial language chosen for development of MicroShell was "C" - the language developed by Bell Labs for writing the UNIX operating system. The Software Tools had been developed in RATFOR, a structured preprocessor for FORTRAN, which resulted in good transportability of the Software Tools from one operating system and computer to another. It was decided that the code generated by RATFOR was too large for the limited CP/M environment. In addition, transportability was a secondary goal. So BDS C was chosen as MicroShell's language.

By June 1981, MicroShell in "C" was up and running and in daily use with CP/M. At 12K, MicroShell still was larger than desired. So a rewrite of portions of MicroShell into assembly language was begun. The current version of MicroShell is about 9.5K bytes. Though slightly larger than we ultimately desire, we believe it presently represents an acceptable tradeoff between capabilities included and memory size requirements.

MicroShell is executed from CP/M by typing "sh". CP/M loads in MicroShell which then relocates itself below CP/M (just below the Basic Disk Operating System - BDOS). MicroShell replaces the CP/M Console Command Processor (CCP) and performs all of the functions of the CP/M CCP plus additional UNIX-shell-like functions. It remains resident during execution of programs until it is deliberately exited by the user. In this respect it is similar to Wordstar and other programs which themselves perform CCP functions while remaining resident.

Appendix B

MicroShell Error Messages

(Filename)?

MicroShell cannot find the file "Filename" to execute along its search path. See Section 3 for an explanation of MicroShell's search path.

Arg > 19

Argument substitution for an argument number greater than the 19 permitted was found in a shell file. Change the shell file to use no more than 19 arguments.

Can't load "sh" from shell

User has attempted to execute MicroShell ("sh.com") from within MicroShell itself. MicroShell has not been extensively tested in this mode and may not handle all conditions properly so it is currently prohibited. An adventuresome user may experiment with running in this mode by placing an explicit drive prefix in the command, e.g.:

```
a:sh
```

One of the things which can be done in this mode (MicroShell running within MicroShell) is to redirect all output of the child shell (the second "sh" invoked). If this is done with the ">+" operator, the result is a file of the entire terminal session of the child shell. So if the prompt is changed to look like the CP/M prompt, this could be used to prepare documentation of what appears to be a CP/M session. There are many possibilities here.

Can`t open input file

Occurs during input redirection. MicroShell can't find the file specified on the command line as the input redirection file.

Can`t open output file

Occurs during output redirection. MicroShell can't open the output redirection file. The directory is probably full; remove some files from the directory and repeat the command.

Disk Full

The disk has filled up while MicroShell is redirecting output (or during the first command in a pipe.) The command is aborted. Make room on the disk for the output redirection file or specify a disk other than the default disk on the command line, e.g. "stat *.* >+a:statout". The disk for the temporary pipe files ("PYPE1", etc.) cannot be specified; it is always the default disk. Use output redirection to a temporary file instead of a pipe if another disk is desired for the temporary file.

End input file: input from console

If a program attempts to read past the physical end of an input file during input redirection from the file (or during the second command in a pipe), MicroShell will issue this message and shift the input source to the console. If this error message occurs, in most cases there's a problem. The file probably has not caused the program reading it to terminate normally. Look at the input file and see if it is supplying the program with enough input to properly cause the program to terminate. (See Section 2.6.3)

Too many characters from input file

When input redirection is in progress (or during the second command in a pipe) for programs which are reading lines of buffered input from a file (BDOS call 10), the calling program establishes a maximum allowable buffer length into which the characters are placed by CP/M. Without input redirection, if the user attempts to enter more characters into the buffer than its maximum size allows, CP/M merely terminates the call for a line of input and returns to the user. This is occasionally seen in DDT when more than 31 characters are typed. During input redirection on buffered input calls, MicroShell monitors the maximum buffer size of the caller to ensure that the buffer is not overflowed. This situation is normally caused by some error that results in a line of input from an input file being too long for the caller's buffer. After sending the error message, MicroShell terminates the current command, any shell file or multiple commands that may be pending and prompts for the next command. Look at the input file for lines longer than the calling program's buffer capacity.

Two output or input files

Two output or input redirection files were specified. The following are examples of illegal commands:

```
% stat *.* >statout >statoutl      (Two output files)
% ed test <script <edscript        (Two input files)
% type script | ed test <script    (Two input files in 2nd
                                    half of command.)
% type script >outfile | ed test   (Two output files in 1st
                                    half of command.)
```

Note: Pipes are actually an output redirection of the first half of the pipe to a temporary file ("PYPE#") followed by an input redirection from that temporary file in the second half of the pipe. An input redirection in the first half or an output redirection (or another pipe) in the second half of the pipe are legal.

CP/M Error Messages

The CP/M Basic Disk Operating System (BDOS) issues various other messages, beginning with:

BDOS ERROR ON (Drive): ....

See the Digital Research CP/M documentation for the meaning of these errors.

APPENDIX C

MicroShell Compatibility with other Programs

Compatible Programs: Many popular CP/M compatible programs have been run under MicroShell with satisfactory operation. The following is a partial list:

1. CP/M utilities: ASM,DDT,ED,LOAD,STAT,PIP,SYSGEN
2. Assemblers: MAC, ACT 80, ACT 86
3. Word Processors: Wordstar/Spellstar, Benchmark, Spellbinder
4. Languages: BDS C, CBASIC II, BASCOM, PASCAL/M
5. Apple CP/M (with Microsoft Softcard)

Incompatible Programs: Programs which are incompatible with MicroShell are usually accessing information inside of CP/M rather than using the normal CP/M entry points. MicroShell depends on a program using the normal entry points for CP/M: the BDOS entry point at location 5 and the warm start entry point at location 0. If a program directly accesses the BIOS jump table, MicroShell will work properly although it will not be able to redirect input or output. If the program changes the BIOS jump table entries, MicroShell may not work properly.

The following programs are known to be incompatible with MicroShell:

1. CP/M utilities:

   MOVCPM - Requires that the CP/M CCP be present. Exit MicroShell to do MOVCPM.

   SUBMIT - MicroShell provides a capability equivalent to the CP/M SUBMIT function. SUBMIT itself requires that the CP/M CCP be present. A useful feature is described in Section 4 for having MicroShell create a submit file and then exit to CP/M to execute the submit file. This permits long programs, which require all of the computer's memory, to be started from MicroShell and MicroShell reloaded automatically on completion of the program.

2. CP/M User Group Programs

   Some of the earlier utilities in the CP/M Users' Group did not access CP/M via its design entry points. Some of these programs may not operate correctly with MicroShell.

Appendix D

BIBLIOGRAPHY

The following books and articles represent a few of the sources of information on the UNIX operating system and its Shell.

Bourne, S. R., "An Introduction to the UNIX Shell." The Bell System Technical Journal 57 (1978):2797-2822.

Gautier, Richard L. Using the UNIX System. Reston Publishing Company, Inc. A Prentice-Hall Company, 1981.

Hall, D. E.; Scherrer, D.K.; and Sventek, J. S. "A Virtual Operating System." In Communications of the ACM 23 (1980):495-502.

Johnson, Stephen C. "UNIX Time-Sharing System: Language Development Tools." Bell System Technical Journal 57 (1978): 1971-90.

Kernighan, Brian W., and Plauger, P. J. Software Tools. Reading, Massachusetts: Addison-Wesley, 1976.

Kernighan, Brian W., and Ritchie, Dennis M. The C Programming Language. Englewood Cliffs: Prentice-Hall, 1978.

Thomas, Rebecca and Yates, Jean A User Guide to the UNIX System. Berkeley, California: OSBORNE/McGraw-Hill, 1982.

Appendix E

Summary of MicroShell Commands

Special MicroShell Characters in Command Line

Char	Meaning	Example
>	Output Redirection	stat >filename
<	Input Redirection	ed file <script
|	Pipe output to input of next cmd ("^" and "|" are equivalent)	prog1 | prog2 | ... stat *.*|pip lst:=con:
^	"^" in shell and input files causes next character to be its control equivalent.	^C (or ^c) changed to 03
:	When first character on a shell file line, causes line to be treated as a comment (ignored).	: this is a comment
+	Echo redirected Output to Console	stat >+filename prog1 |+ prog2 | ...
-	Return "character ready" to console input status calls	sysgen <-script prog1 |- prog2 | ... prog1 |+- prog2 | ...
;	Separate commands	era *.bak;stat;ed test <script
"	Treat arguments with embedded spaces or tabs as 1 argument and ignore special characters inside quotes	echo "This is one argument"
\	Ignore special meaning of next character	dir \>file (filename ">file")
$	Argument substitution in shell (command) files (0-19)	comfile test data if "comfile.sub" contains: pip b:=a:$1 pip b:=a:$2 then MicroShell executes: pip b:=a:test pip b:=a:data
$T	Redirect Input back to console in a command file	pip b:=a:$1 b:=a:$2 ddt <$T

Shell Flags

Flag	Meaning	Example
+f or +F (Default)	Auxiliary file search enable	% +F (Auxiliary file search on)
-f or -F	Auxiliary file search disable	% -F (Auxiliary file search off)
+g or +G (Default)	Gobble line feeds during Input redirection	% +G (Line feeds removed from input)
-g or -G	Don't gobble line feeds during Input Redirection	% -G
-l or -L or +l or +L	Login current disk (after changing disks)	% -l % (new disk logged into CP/M for writing)
-p or -P or	Prompt string Uses "C"-like format: % - Next char special (%% gives %)	-p "%n%%" gives: (CR, LF)%
+p or +P	n/N - Newline (CR, LF) d - Lower case drive D - Upper case drive u/U - User number	-p "%nDrive:%D User%U %" gives: (CR, LF)Drive:A User:0 %
-s or -S or +s or +S	Shell Status report (Shows status of flags)	% -S File search: On Gobble lfs: Off Ucase cmd: On Verbose: Off
+u or +U (Default)	Upper case translation of command line (like CP/M)	% +U % echo this is upper case THIS IS UPPER CASE
-u or -U	No case translation on command line (allows passing lower case command line to a program)	% -U % echo this is lower case this is lower case
+v or +V	Verbose mode: Echo commands before execution	% +V (All commands echoed) comfile test data pip b:=a:test pip b:=a:data
-v or -V (Default)	Disable Verbose mode	% -V (No echo of commands)
-x or -X	Exit MicroShell and return to CP/M	% -x A>

APPENDIX F

INDEX